HSM-based Key Management Solution for Ethereum Blockchain

Wazen M. Shbair [*], Eugene Gavrilov [†], Radu State [*]
[*] University of Luxembourg, SnT, 29, Avenue J.F Kennedy, L-1855 Luxembourg
Email:{wazen.shbair, radu.state}@uni.lu
[†] VNX Exchange, Luxembourg
Email: eugene.gavrilov@vnx.io

***Abstract*—The security of distributed applications backed by blockchain technology relies mainly on keeping the associated cryptographic keys (i.e. private keys) in well-protected storage. Since they are the unique proof of ownership of the underlying digital assets. If the keys are stolen or lost, there is no way to recover the assets. The cold wallet is a good candidate for basic use cases, but it has a substantial challenge for more complex applications as it does not scale. Warm and hot wallets are more convenient options for blockchain-based solutions that aim to transact in a cloud environment. In this work, we focus on Hardware Security Module (HSM) based wallet. The HSM is the de-facto standard device designed to manage high-value cryptographic keys and to protect them against hacks. In this demonstration, we present an HSM-based working prototype that secures the entire life cycle of Ethereum public and private keys.**

I. Introduction

Blockchain technology shows promise in its ability to create new financial trading, payments solution, business model, and wide range of applications. However, recent high-profile breaches of exchanges reveal that the storage of users' cryptographic keys presents a security weakness that must be resolved properly. In blockchain-based applications, the associated cryptographic keys are the unique proof of ownership of the underlying digital assets. Therefore, no one can steal the digital asset unless he/she has access to the private key to sign spending or transferring transactions. Even worse if the access to the private key is lost then the fund will be locked out and no way to access it at all [1].

Due to lack of proper key management solutions up to 23% of Bitcoins - currently worth around $140 billion - cannot be accessed as a result of lost or forgotten keys. Therefore, the management of cryptographic keys is one of the most critical and challenging components of the cryptographic system. No blockchain-based platform is secure if its keys management module is not well-secured [2].

The cryptocurrency wallet is a means to store private keys securely, so they are accessible only with the right permission. Digital assets or cryptocurrencies are not stored as physical fiat money within the wallet, thus to make a transaction a user retrieves a private key from secure storage and then signs the spending transaction using the private key. Once the signed transaction is broadcasted to the selected blockchain platform the fund is released and stored in an immutable manner.

Various cryptocurrency wallet solutions are ranging from paper-based to cloud-based wallets. Classifying the wallets is varying based on various criteria. For instance, based on the Internet connectivity we find Cold, Warm, and Hot wallets [3]. Based on the physical status we find Software and Hardware and Paper wallets. While based on the secure trusted environment technology we have the Trusted Execution Environment (TEE) and Hardware Security Module (HSM) wallets. A comprehensive benchmarking of cryptocurrency wallets can be found in [4].

Hereby, we focus on HSM-based wallets. HSM is a well-known solution in the banking industry since it provides a secure environment for transaction processing and user's Personal Identification Number (PIN) verification. Via strong cryptographic algorithms, the HSM device generates and stores keys, ensuring that the master key never leaves the vault. Also, the HSM hardware allows executing cryptographic operations in a trusted environment. On top of that, the HSM device is equipped with a fully self-protecting circuit, so if tamper sensors detect a possible attack, all critical keys are immediately destroyed and the HSM device becomes permanently inoperable [5].

In this demonstration, we will present a key management solution based on the HSM technology to control the life cycle of Ethereum public and private keys. Alongside the cryptographic operations required for signing transactions.

II. Design and Implementation

HSM-based solutions are designed and certified to provide the highest level of physical security. In this context, HSM devices can be used to leverage this established hardware technology to foster the security of digital wallets in the blockchain ecosystem. This section details the core functions of generating Ethereum wallets and calculating valid signatures. Due to the high cost of HSM services our prototype has been implemented using SoftHSM - a software emulation framework for the HSM hardware [6].

A. Ethereum Address Generation

According to Ethereum Yellow paper [7] for a given Elliptic Curve Digital Signature Algorithm (ECDSA) private key Pr, the corresponding Ethereum address $A(Pr)$ is defined as the rightmost 160-bits of the Keccak-256 hash of the corresponding ECDSA public key [8]. Algorithm 1 provides the required steps to generate an Ethereum wallet based on a given ECDSA key pair, where the public key will be used to calculate the related Ethereum address, and the private key will be used for signing transactions. Thanks to the HSM module that will generate and store the required ECDSA key pair.

Algorithm 1: Ethereum wallet generation

```
Input: SoftHSM Module ($mod$)
Output: Ethereum wallet address
1 Function Generate_Ethereum_Wallet($mod$):
2     $keys \leftarrow hsm\_session.generateKeyPair(ecdsa)$
3     $publicKey \leftarrow keys.publicKey$
4     $address\_hash \leftarrow keccak256(publicKey)$
5     $eth\_Addr \leftarrow "0x" + address\_hash.slice(-20)$
6     return $eth\_Addr$
```

B. ECDSA Signature Calculation

The ECDSA signing algorithm typically takes as input a message to be signed and a private key. As output it returns a signature presented by a pair of 256-bit integers identified by R and S [9]. As a special case, the Ethereum blockchain add an additional V value, named recovery identifier. Therefore the Ethereum transactions signature is notated as R, S, V. The signature can be presented as one 65-byte-long sequence; 32 bytes for R, 32 bytes for S, and one byte for V (27 or 28). The V value is essential since we are working with elliptic curves (i.e ECDSA), multiple points on the curve can be calculated from R and S alone. This may give two different public keys that can be recovered. The V identifier indicates which one of these points to use. Even more, sometimes the S value can randomly be on the 'wrong side' on the ECDSA curve. So we need to keep signing until we get a good value of S as detailed in Algorithm 2.

C. Ethereum Transaction Signature

The ECDSA signature assigned to an Ethereum transaction proves that the sender of the transaction had the required access to the private key, and the transaction has not been changed since it was signed. Therefore, two pieces of information should be signed; the sender's address and the transaction object. To sign the spending address we pass it to Algorithm 2 with the right private key. The returned values of R, S, V will be used as parameters in the transaction object. Then the raw transaction object is signed also using the later Algorithm, where the returned R, S, V present the required signature component for the transaction to be submitted and executed.

D. Prototype Implementation and Limitation

Our prototype[1] has been developed using SoftHSM and Web3.js [10]. The prototype comprises Ethereum wallets generation and storage, besides transactions building and signing

[1]Publicly available on https://github.com/wshbair/HSM2ETH

Algorithm 2: ECDSA signature calculation

```
Input: Message $msg\_to\_sign$, PublicKey$Pk$,
       PrivateKey$Pr$
Output: Signature components $R, S, V$
1 Function
  Calculate_ECDSA_Signature($msg\_to\_sign, Pr$):
2     $msg\_hash \leftarrow keccak256(msg\_to\_sign)$
3     $flag \leftarrow True$
4     while $flag$ do
5         $signature \leftarrow$
            $hsm\_session.sign(msg\_hash, Pr)$
6         //Extract the $S$ value from the $signature$
7         $S = signature.slice(32, 64)$
8         if $S < (secp256k1.size/2)$ then
9             $flag \leftarrow False$
10    $R \leftarrow signature.slice(0, 32)$
11    $S \leftarrow signature.slice(32, 64)$
12    $V \leftarrow 27$
13    $recovered\_key \leftarrow$
        $recover\_ecdsa\_publicKey(R, S, V)$ if
        $recovered\_key \neq Pk$ then
14        $V \leftarrow 28$
15    return $R, S, V$;
```

functions. Ethereum Rinkeby testnet has been used to validate the generated and signed transactions.

Our solution, however, is subject to a potential limitation in terms of scalability. Because all generated keys are stored inside the HSM device. However, HSM devices have a finite number of "slots", and each slot has a finite number of keys. For instance, AWS CloudHSM can store only 3300 keys [11].

To solve this, we can generate a master key using the HSM module. The master key will be used to encrypt the seed phrase of an Ethereum wallet. A seed phrase is a way to access and recover a digital wallet. It contains a random sequence of words, usually 12 or 24 English words. The sequence is converted using formulas to numbers that give access to the corresponding digital wallet and the public and private keys. In this case, we can store locally the encrypted seed, the master key, and the generated Ethereum address. Thus, to transact on behalf of the Ethereum address we follow these steps: (1) decrypt the seed using the master key; (2) regenerate the wallet (i.e public and private kes) using the decrypted seed; (3) build and sign the transaction using the regenerated wallet.

As future work, we intend to stretch this work to cover the key management for different cryptocurrencies such as Bitcoin, Ripple XRP, and Steller. Also, we will evaluate our approach using well-known HSM service providers.

III. Acknowledgements

This work has received partially funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 830927, the CONCORDIA project.

REFERENCES

[1] O. Boireau, “Securing the blockchain against hackers,” *Network Security*, vol. 2018, no. 1, pp. 8–11, 2018.

[2] O. Pal, B. Alam, V. Thakur, and S. Singh, “Key management for blockchain technology,” *ICT Express*, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405959519301894

[3] H. Rezaeighaleh and C. C. Zou, “New secure approach to backup cryptocurrency wallets,” in *IEEE Global Communications Conference-Communication & Information Systems Security Symposium*, 2019.

[4] G. Hileman and M. Rauchs, “2017 global cryptocurrency benchmarking study,” *Available at SSRN 2965436*, 2017.

[5] R. Focardi and F. L. Luccio, “Secure upgrade of hardware security modules in bank networks,” in *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*. Springer, 2010, pp. 95–110.

[6] “Opendnssec softhsm,” https://opendnssec.org/softhsm, [Online; accessed 6-Jan-2021].

[7] G. Wood *et al.*, “Ethereum: A secure decentralised generalised transaction ledger,” *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[8] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “Keccak,” in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2013, pp. 313–314.

[9] T. Pornin, “Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa),” *Internet Engineering Task Force RFC*, vol. 6979, pp. 1–79, 2013.

[10] “web3.js - ethereum javascript api,” https://github.com/ethereum/web3.js, [Online; accessed 7-Mar-2021].

[11] “Aws cloudhsm - user guide,” https://docs.aws.amazon.com/cloudhsm/latest/userguide/cloudhsm-user-guide.pdf, [Online; accessed 7-Mar-2021].